Rafiev A, Xia F, Iliasov A, Gensh R, Aalsaud A, Romanovsky A, Yakovlev A. Order Graphs and Cross-Layer Parametric Significance-driven Modelling. *In: 15th International Conference on Application of Concurrency to System Design (ACSD'15)*. 2015, Brussels, Belgium: IEEE Computer Society.

Order Graphs and Cross-layer Parametric Significance-driven Modelling

A. Rafiev, F. Xia, A. Iliasov, R. Gensh, A. Aalsaud,
A. Romanovsky, A. Yakovlev – Newcastle University, UK
{ashur.rafiev, fei.xia, alexei.iliasov, rem.gensh, a.m.m.aalsaud,
alexander.romanovsky, alex.yakovlev}@ncl.ac.uk

***Abstract*—Traditional hierarchical modelling methods tend to have layers of abstraction corresponding to naturally existing layers of concern in multi-level systems. Although logically and functionally intuitive, this is not always optimal for analysis and design. For instance, parts of a system in the same logical layer may not contribute to the same degree on some metric, e.g. system power consumption. When focusing on a specific parameter or set of parameters, to moderate the analysis, design and runtime effort, less significant parts of the system should be modelled at higher levels of abstraction and more significant ones with more detail. This parametric significance-driven modelling approach focuses more on optimal parametric fidelity than on logical intuition. Using system power consumption as an example parameter, this paper presents Order Graphs (OGs), which have a clear hierarchical structure, and provide straightforward vertical zooming across multiple layers (orders) of model abstraction, resulting in the discovery of power-proportional cuts that run through different orders to be analysed together in a flat manner. Stochastic Activity Networks (SANs), a good flat modelling method, is suggested as an example of studying techniques for cuts discovered with OGs. A series of experiments on an Odroid development system consisting of an ARM big.LITTLE multi-core structure provides initial validation for the approach.**

I. Introduction

Systems with large scale concurrency and complexity, e.g. computation systems built upon architectures with multiple and increasingly many processing cores with heterogeneity among the components, are becoming more popular and common-place [3]. The hardware motivations are clear, as concurrency scaling can help delay the potential saturation of Moore's Law with current and future CMOS technology and better use the opportunities provided by the technology scaling. In this environment, software designs are increasingly focused towards greater concurrency and mapping to such many-core hardware [12].

Both hardware and software of these types tend to form hierarchical structures, for instance, the levels of detail in hardware include the entire spectrum from transistors to gates to function blocks to entire CPUs to multiple CPUs with supporting logic, memory, etc. For system designers, software (e.g. applications), operating systems, and the platforms on which these are run also form natural design layers with clear boundaries between the layers. Such structures are usually conveniently modelled with traditional hierarchical modelling methods, with the modelling levels of abstraction corresponding to these system layers of concern [10].

This is, however, not always optimal for analysis, design and runtime management, which functionally and pragmatically usually focus on a set of physical parameters connected to the notion of 'performance' known as system metrics. The "modelling fidelity" should therefore, ideally, be determined by the parameter(s) under study [19], [16]. For instance, when designing for power efficiency, if a part of a system makes a crucial contribution to the power consumption of the entire system and small changes may have a significant effect, it pays to study it in detail, i.e. at some lower layer of abstraction. On the other hand, to moderate the modelling, analysis and design effort, and potentially runtime overhead for models that need to be used in runtime, other less significant parts of the system should be studied at higher levels of abstraction. When this "centre of gravity" of system operation concerning power (and/or other important parameters) can dynamically move around the system, traditional hierarchical modelling methods are ill positioned for efficient representation.

Hierarchical methods, because of their complexity, are usually less straightforward to use than flat representations. Petri nets [2], which exemplify flat modelling methods, have extremely simple semantics and offer conveniences in reasoning, proofing and other aspects of analysis, a quality shared by other flat modelling methods. But when the modelling needs span multiple layers in a hierarchy it becomes somewhat difficult to adopt flat methods as study tools.

In this paper, we present Order Graphs (OGs), a model that has a clear hierarchical structure, and at the same time provides straightforward vertical zooming across multiple layers (orders) of model abstraction, independently in different regions of a model, resulting in the isolation of cuts that run through different orders for reasons important for the designer. For instance, one of these reasons may be to identify cuts with a constant fidelity for some parameter. Such cuts can then be analysed in a flat manner using existing flat modelling methods. The concept not only works during design time but are also applicable at runtime. Parametric significance-driven cuts can move with the system's operation based on the changes of the significance of parameters related to different system parts and thus always concentrate the most appropriate amount of modelling effort on each part of the system resulting in the optimal modelling fidelity for each part at each state. The ideal scenario will lead to true parametric-proportional modelling fidelity and thus parametric-proportional effort for analysis, design, and runtime management.

In this paper, the usefulness of parametric-proportional cuts, obtained with the help of OGs through cross-layer reasoning, is demonstrated by using Stochastic Activity Networks (SANs) [17] to study such cuts. Although other flat modelling methods can also be used for such studies, SANs, a quantitative derivative of Petri nets with both probabilistic and deterministic representation facilities, have the advantage of being useful for reasoning about qualitative issues such as correctness and quantitative issues such as parameter values. A series of experiments on an Odroid development system consisting of an ARM big.LITTLE multi-core structure [9] provides initial validation for the approach.

The initial concept of this work has been introduced in [16]. This paper presents the formal definition of OGs compared to classic hierarchical representation of graphs, which includes the theory on model transformations and on forming cross-layer cuts, and gives more detailed insight on the benefits of using cross-layer cuts w.r.t. the state space size.

Section II formally describes the modelling approach including resource graphs, hierarchical graphs, and OGs. Section III describes the overall parametric significance-driven modelling flow. Section IV presents the application example Odroid system with platform modelling in OGs and component modelling in SANs, and a series of experiments which support the approach. Section V includes discussions. In this paper we take system power consumption as the example parameter and focus on power-proportionality in modelling, but it must be emphasised that the method itself is parameter-independent.

II. Modelling Approach

A. Resource-driven Modelling

The central subject of our method is the study of a computational platform comprising a number of diverse resources and the way resources may be handled in order to realise a computation. A resource is in this case an indivisible element required by the system in order to change its state, and it is defined by its function and availability in relation to this transition. With the word "resources" we make the point that we do not exclude computation, communication, or other facilities, e.g. energy and time. A resource graph is a relation graph, where each vertex represents a single resource and each edge represents a relation or dependency between two resources [15].

In many real-life systems the dependencies between the resources do not have to be maintained all the time in order for the system to function normally. In fact, for some systems the functionality requires switching dependencies on and off [20]. In this paper, however, we focus on a static resource view of the system, showing all possible resources and dependencies. This way of modelling is focused on exploring the structure of the system and does not provide the means to estimate quantitative properties of the system. For quantitative analysis we can use other methods, for example, SANs [17].

Focusing on resources in modelling is a well-established practice. For example, resource dependency graphs [4], [14] are partial orders representing causality relationships between resources. In our case, however, the view on resource dependencies is focused on a pair of resources being dependent on each other at a certain point in time for delivering their functionalities. Hence, there is no directionality, and dependencies are dynamic.

In our previous work [15], approaching the cross-layer modelling was suggested using labelling in the flat resource graphs. However, it appeared to be impractical compared to hierarchical models. The following discussion revisits the definition of a hierarchy as a sequence of model transformations, which thereafter is applied to graph models leading to Order Graphs. The latter combines the notions of resource modelling with the hierarchical representation of system layers.

B. Introducing Hierarchies

An underlying approach for having adjustable fidelity in the models relies on different levels of abstraction. Naturally, these layers have to be consistent with each other, however the very definition of consistency may vary from model to model and depend on the system properties that need to be preserved.

A common way to define a model of a system is to represent it as a set tuple $M = (E_1, E_2, \dots E_n)$, where each set E_k contains system elements of a particular type, e.g. vertices, edges, labels, etc. We can also generalise these to a single type – "system element" – and have a type-agnostic representation of a model:

$$\mathcal{M} = E_0 \cup E_1 \cup \ldots \cup E_n. \tag{1}$$

Definition 1. Let M_a and M_b be some system models with corresponding sets of system elements $\mathcal{M}_a, \mathcal{M}_b$, and some relation between these elements $\gamma \subseteq \mathcal{M}_a \times \mathcal{M}_b$. Given a boolean predicate Φ, such that:

$$\Phi : \mathbb{P}(M_a) \times \mathbb{P}(M_b) \times \mathbb{P}(\mathcal{M}_a \times \mathcal{M}_b) \to \{0, 1\}, \tag{2}$$

the relation γ is called a *consistency relation* between models M_a and M_b under the predicate Φ if $\Phi(M_a, M_b, \gamma) = 1$. Φ is called the *rule set*, and for convenience can be specified as a conjunction of smaller predicates of the same type (2).

The predicate Φ is called *strongly consistent* if it requires γ to be a total surjective relation, i.e. for every element in $\mathcal{M}_a$ there must be at least one related element in $\mathcal{M}_b$, and for every element in $\mathcal{M}_b$ there must be at least one related element in $\mathcal{M}_a$. In this case, γ is called a *transformation*; transformations are further denoted as $\gamma = \mathcal{M}_a \vdash \mathcal{M}_b$ (or $\gamma = M_a \vdash M_b$ since $\mathcal{M}_a, \mathcal{M}_b$ are derived from M_a and M_b).

Definition 2. Let $\{\ldots, M^{(k-1)}, M^{(k)}, M^{(k+1)}, \ldots\}$ be an infinite or finite set of models of the same system, where each $M^{(k)}$ models the system in a specific level of details. An *abstraction hierarchy* is a total order of models where any two adjacent models form a transformation $\gamma_k = M^{(k)} \vdash M^{(k+1)}$ under a given strongly consistent predicate Φ_k, and the size of models monotonically decreases (or increases) with k:

$$\mathcal{H} = \ldots \vdash M^{(k-1)} \vdash M^{(k)} \vdash M^{(k+1)} \vdash \ldots \tag{3}$$

Each $M^{(k)}$ is k-th level of abstraction, also called *order* k.

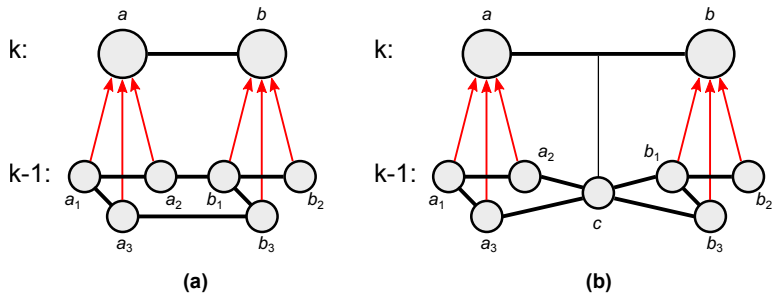

Figure 1. Conventional hierarchy representation (a) compared to Order Graphs (b); k is the higher level of abstraction and $k-1$ is the lower level.

A hierarchy is called *homogeneous* if it uses the same rule set Φ for all its consistency relations; this implies that $\mathbb{P}\left(M^{(k)}\right) = \mathbb{P}\left(M^{(k+1)}\right)$ for all k.

Each hierarchy contains both horizontal and vertical knowledge: each abstraction layer $M^{(k)}$ is a horizontal view of the system, while the set of relations $\{\ldots, \gamma_k, \gamma_{k+1}, \ldots\}$ stores the information on how different layers interlink. Notions of horizontality and verticality have been first introduced in [8].

C. Hierarchical Graphs

Figure 1(a) shows the conventional approach to hierarchical graphs, which is based on clustering and uses tree structures [10]. Each node of the higher layer zooms into a subgraph in the lower layer. Consequently, an edge between two nodes becomes multiple edges between the corresponding subgraphs. The notation used in the diagram is based on Zoom Structures [8]. A convenient way to display graph hierarchies is zoom views, showing verticality and horizontality with vertical and horizontal arcs respectively. The following is a redefinition of hierarchical graphs in the terms presented in Section II-B.

Definition 3. *Hierarchical graph* is a homogeneous hierarchy, such that, each k-th order is a graph $G^{(k)} = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges; and all consistency relations in this hierarchy are defined as follows:

$$\gamma = \gamma_v \cup \gamma_{v+} \cup \gamma_e.$$

Let $G^{(k)} = (V, E)$ and $G^{(k+1)} = (V', E')$, called lower and higher orders respectively. Thus, $\gamma \subseteq (V \cup E) \times (V' \cup E')$.

Inclusion function $\gamma_v : V \to V'$ represents vertex clustering.

Supplementary inclusion function $\gamma_{v+} : E \to V'$ and predicate (4) ensure that all edges within a cluster are also included.

$$\forall \langle v_1, v_2 \rangle \in E, v' \in V' : \gamma_{v+}\left(\langle v_1, v_2 \rangle\right) = v' \Leftrightarrow \\ \Leftrightarrow \gamma_v\left(v_1\right) = v' \wedge \gamma_v\left(v_2\right) = v'. \quad (4)$$

Edge grouping function $\gamma_e : E \to E'$ groups edges connecting vertex clusters:

$$\forall \langle v_1, v_2 \rangle \in E, \langle v_1', v_2' \rangle \in E' : \gamma_e\left(\langle v_1, v_2 \rangle\right) = \langle v_1', v_2' \rangle \Leftrightarrow \\ \Leftrightarrow \gamma_v\left(v_1\right) = v_1' \wedge \gamma_v\left(v_2\right) = v_2'. \quad (5)$$

Thus, inclusion function γ_v can be chosen arbitrarily; and from it, the predicates (4) and (5) uniquely describe the edges in the hierarchical graph.

The most important property of the rule set defined above is that it preserves all paths in the graph during the mapping. In other words, for any vertices $v_1, v_2 \in V$ and related vertices $v_1', v_2' \in V'$, if there exists a path between v_1 and v_2 in $G^{(k)}$, there will be a path between v_1' and v_2' in $G^{(k+1)}$, and vice versa:

$$\forall v_1, v_2 \in V, v_1', v_2' \in V' : \gamma_v\left(v_1\right) = v_1' \wedge \gamma_v\left(v_2\right) = v_2' \Rightarrow \\ \Rightarrow \left(P\left(v_1, v_2\right) \Leftrightarrow P\left(v_1', v_2'\right)\right), \quad (6)$$

where $P\left(x, y\right)$ is a function that is true iff there is a path between x and y.

D. Order Graphs

By the nature of resource graphs, according to Section II-A, anything can be considered a resource. Can we say that the edges of a graph are also resources? It is actually true, and this contradiction is explained and solved by Order Graphs.

As an example, let's imagine that Figure 1(a) models a network interaction, where a is a server and b is a client. On the very abstract level we do not care about the structure of the network at this level of abstraction, we just need to know that the client and the server are connected somehow, thus we model this entire system as the client and the server connected directly with a single dependency. However, in a detailed model we can no longer ignore the network protocols and have to introduce it at least as a single resource node as shown in Figure 1(b).

A distinct property of the proposed OG modelling approach is that a high-order edge relates to a node at the lower order. In this case we say that the node *supports* an edge, while in fact this is the same entity viewed from the different abstraction levels. In real-life systems, any dependency is always supported by a resource of some kind, and this "fractal" structure goes down to the smallest details, including atoms and below. Of course, we do not want to include all these in the model, so we had to flex the rule by saying that an edge is either supported by a resource at the lower layer or stays an edge like in conventional hierarchical graphs.

Definition 4. *Order Graph* is a homogeneous hierarchy, such that, each k-th order is a graph $G^{(k)} = (V, E)$, where V is

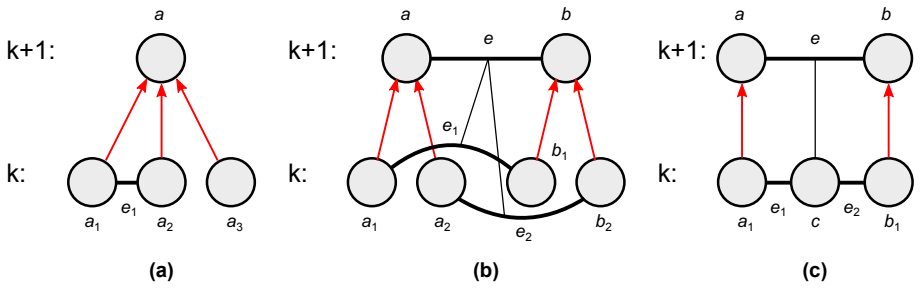

Figure 2. Elementary transformations in Order Graphs and their notation: (a) inclusion, (b) edge grouping, (c) support.

the set of vertices and $E \subseteq V \times V$ is the set of edges; and all consistency relations in this hierarchy are defined as follows:

$$\gamma = \gamma_v \cup \gamma_{v+} \cup \gamma_e \cup \gamma_s \cup \gamma_{s+}.$$

Here $\gamma_v, \gamma_{v+}, \gamma_e$ are defined as in Section II-C.

Let $G^{(k)} = (V, E)$ and $G^{(k+1)} = (V', E')$; *support* function $\gamma_s : V \leftrightarrow E'$ is a one-to-one mapping of some vertices onto some of the edges of a higher order graph.

The first rule on γ_s is as follows:

$$\begin{aligned}\forall v \in V, \langle v_1', v_2' \rangle \in E' : \gamma_s(v) = \langle v_1', v_2' \rangle \Rightarrow \\ \Rightarrow (\exists v_1 : \langle v_1, v \rangle \in E \wedge \gamma_v(v_1) = v_1') \wedge \\ \wedge (\exists v_2 : \langle v, v_2 \rangle \in E \wedge \gamma_v(v_2) = v_2'), \quad (7)\end{aligned}$$

meaning that we can map vertex v onto an edge $\langle v_1', v_2' \rangle$ iff v is connected to at least one vertex related to v_1' and at least one vertex related to v_2'. In addition, all vertices adjacent to v must be related either to v_1' or v_2':

$$\begin{aligned}\forall v, v_{adj} \in V, \langle v_1', v_2' \rangle \in E' : \gamma_s(v) = \langle v_1', v_2' \rangle \Rightarrow \\ \Rightarrow (\langle v_{adj}, v \rangle \in E \wedge \gamma_v(v_{adj}) = v_1') \vee \\ \vee (\langle v, v_{adj} \rangle \in E \wedge \gamma_v(v_{adj}) = v_2'). \quad (8)\end{aligned}$$

Finally, the same vertex cannot be used in a vertex-to-vertex and a vertex-to-edge relation; and the same higher order edge cannot be used in an edge-to-edge and a vertex-to-edge relation:

$$\begin{aligned}\mathrm{dom}\gamma_v \cap \mathrm{dom}\gamma_s &= \emptyset, \\ \mathrm{ran}\gamma_e \cap \mathrm{ran}\gamma_s &= \emptyset.\end{aligned}$$

Supplementary support function $\gamma_{s+} : E \to E'$ groups all edges adjacent to v into the same higher order edge using the following predicate:

$$\begin{aligned}\forall \langle v_1, v_2 \rangle \in E, \langle v_1', v_2' \rangle \in E' : \gamma_{s+}(\langle v_1, v_2 \rangle) = \langle v_1', v_2' \rangle \Leftrightarrow \\ \Leftrightarrow \gamma_s(v_1) = \langle v_1', v_2' \rangle \vee \gamma_s(v_2) = \langle v_1', v_2' \rangle. \quad (9)\end{aligned}$$

OGs preserve paths just like hierarchical graphs (6).

E. Cross-layer Cuts

In the presented work, the analysis of the system is performed on a flat model, not the entire hierarchy. The actual benefit of using hierarchies in this case is in the possibility to obtain flat model (or models) by cutting the hierarchy not horizontally but across layers. The level of details is selected per element of the system, which gives high control on adjusting the precision of the obtained models, ultimately leading to the best size models for the given fidelity requirement.

Let $G^{(k)} = (V, E)$ and $G^{(k+1)} = (V', E')$ be two subsequent orders of an OG. For some subgraphs $g = (V_{eq}, E_{eq}) \subseteq (V, E)$ and $g' = (V_{eq}', E_{eq}') \subseteq (V', E')$, we say $g = g'$ iff conditions (10) and (11) are met; hence g is the part of the graph (V, E) that is not changed during the transformation.

$$\begin{aligned}\forall \langle x, x' \rangle \in \gamma : \left(x \in V_{eq} \Leftrightarrow x' \in V_{eq}'\right) \wedge \\ \wedge \left(x \in E_{eq} \Leftrightarrow x' \in E_{eq}'\right), \quad (10)\end{aligned}$$

$$\begin{aligned}\forall \langle v_1, v_2 \rangle \in E_{eq}, \langle v_1', v_2' \rangle \in E_{eq}' : \gamma(\langle v_1, v_2 \rangle) = \langle v_1', v_2' \rangle \Leftrightarrow \\ \Leftrightarrow \gamma(v_1) = v_1' \wedge \gamma(v_2) = v_2', \quad (11)\end{aligned}$$

where $\gamma = G^{(k)} \vdash G^{(k+1)}$ is the transformation between these orders. If such pair g, g' exists, then $G^{(k)}$ is *partially equal* to $G^{(k+1)}$.

Elementary transformation is the minimum set of changes that may happen between two graphs without violating the rule set of OGs. Thus, OGs have the following types of elementary transformations, shown in Figure 2:

- *Inclusion:* Vertices and edges of the lower order are mapped into a single vertex in the higher order. Figure 2(a) shows vertices a_1, a_2, a_3, and edge e_1 being mapped into vertex a; relation $\langle e_1, a \rangle$ is implied from (4) and not drawn. This elementary transformation also appears in conventional hierarchical graphs.
- *Edge grouping:* Edges of the lower order are mapped into a single edge in the higher order. Figure 2(a) shows edges e_1, e_2 being mapped into edge e. The relations are drawn as thin black lines to be differentiated from vertex-to-vertex relations. This elementary transformation also appears in conventional hierarchical graphs.
- *Support:* One vertex is mapped into one edge in the higher order. Figure 2(c) shows vertex c being mapped into edge e; relations $\langle e_1, e \rangle, \langle e_2, e \rangle$ are implied from (9) and not drawn. This elementary transformation is unique to OGs.

Any transformation $\gamma = G^{(k)} \vdash G^{(k+1)}$ in OG can be represented with a sequence of elementary transformations $\gamma = \gamma_1 \circ \ldots \circ \gamma_n$, or:

$$G^{(k)} \vdash G^{(x_1)} \vdash \ldots \vdash G^{(x_n)} \vdash G^{(k+1)}. \quad (12)$$

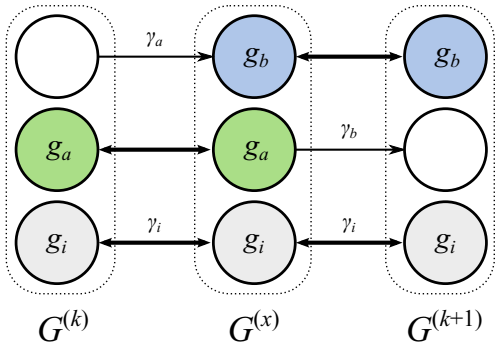

Figure 3. Cross-layer cut $G^{(x)}$ explained.

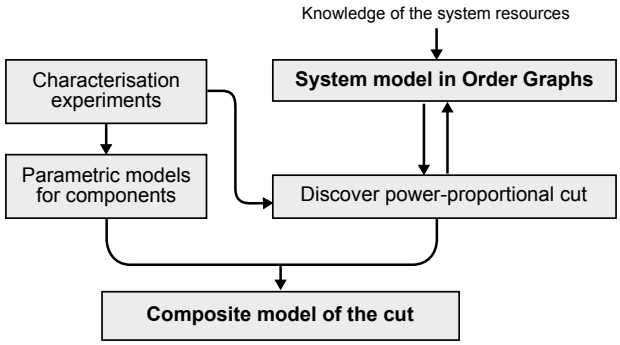

Figure 4. Modelling workflow.

Definition 5. For two subsequent orders $G^{(k)}, G^{(k+1)}$ of an OG, a cross-layer cut $G^{(x)}$ between order k and order $(k+1)$ is a graph, such that $G^{(k)} \vdash G^{(x)} \vdash G^{(k+1)}$ under the same rule set, and $G^{(x)}$ is partially equal to $G^{(k)}$ and $G^{(k+1)}$.

Figure 3 explains the above definition. Let's represent $\gamma = G^{(k)} \vdash G^{(k+1)}$ as $\gamma_a \cup \gamma_b \cup \gamma_i$, such that $\mathrm{dom}\gamma_i = \mathrm{ran}\gamma_i = g_i$ is the unchanged part of the graph; γ_a, γ_b can be chosen arbitrarily. Subgraph $g_b = \mathrm{ran}\gamma_a$ is the part that has been already transformed by γ, one can see that $g_b \subseteq G^{(k+1)}$. Similarly, $g_a = \mathrm{dom}\gamma_b$ is the part that has not been transformed yet, and $g_a \subseteq G^{(k)}$. Therefore, cross-layer cut $G^{(x)} = g_a \cup g_b \cup g_i = \mathrm{ran}\gamma_a \cup \mathrm{dom}\gamma_b \cup g_i$ contains parts from both order k and order $(k+1)$. Note that it is not possible to construct γ_a or γ_b without the other being empty if γ is an elementary transformation. From (12) one can see that a cross-layer cut between $G^{(k)}$ and $G^{(k+1)}$ is one of the steps in a sequence of elementary transformations.

As an example, two possible cuts in Figure 1(b) are $V = \{a_1, a_2, a_3, c, b\}$ (with corresponding edges) and $V = \{a, c, b_1, b_2, b_3\}$ (also, with corresponding edges).

Making a cut through more than two layers – from $G^{(k)}$ to some $G^{(k+b)}$ – can be done iteratively. Firstly, obtain a cut between $G^{(k)}, G^{(k+1)}$, so $G^{(k)} \vdash G^{(x_1)} \vdash G^{(k+1)}$. Then, obtain a cut $G^{(x_2)}$ between newly created $G^{(x_1)}$ and $G^{(k+2)}$, which may now contain parts from $G^{(k)}, G^{(k+1)}$ and $G^{(k+2)}$. Repeat the process until the final cut $G^{(x_{b-1})} \vdash G^{(x_b)} \vdash G^{(k+b)}$ is found.

Cross-layer cuts are models of the same system and are consistent with the layers in the corresponding OG. This consistency preserves graph properties like connectivity, which are used when parametric modelling is applied during the workflow, described in the following section.

III. Modelling flow

We propose a modelling flow based on the exploration of power-proportional cuts using OGs. This modelling flow will result in both fidelity and effort to be as proportional to power as possible (or, instead of power, any metric or combination of metrics). The flow is shown in Figure 4. Characterisation experiments on the components of the system provide information on the crucial parameters and establish the elementary model for each component. The parameters would help to identify appropriate cuts in the OG model for further analysis. The flat model used for this analysis is obtained from composing separate elementary models according to the OG cut.

The method of using hierarchy cuts is non-destructive, meaning that the model can be easily re-arranged and adapted for a different fidelity distribution. This is especially helpful if the model is used in runtime management of the systems working in various "modes". Hence the proposed approach appears advantageous compared to a single flat model.

Intuitively, when using this parametric significance-driven modelling approach, if there are multiple parameters, there is no guarantee that the designer or the runtime management will end up with the same cut for different parameters. This method does not preclude the use of different cuts for different parameters either during design time or runtime. Cut discovery across the fundamental OG system model facilitates the use of different cuts for different parameters in the same way as different cuts for the same parameter at different system states.

At design time, the development of a system OG model can also be a cumulative process of building parametric-proportional cuts. The designer does not have to have a fully detailed OG right from the start. OG elements can be added to models as parameter investigations become more detailed and more and finer characterisation data becomes available. With further research and development of the method, especially with OG-related tools, it is conceivable that this will become true at runtime as well.

The method provides modelling effort savings through two provisions. First the complexity of the overall system OG model may grow in an effort-optimal manner according to parametric design goals because its construction process can be an accumulation of parametric proportional cuts. And second when a complex OG model is already available, at any one time during analysis or runtime, only a specific parametric proportional cut or set of such cuts need to be studied and not the entire detailed system model.

In Section IV we apply this flow to obtain power-proportional models for an example platform. We use SANs as a possible technique for quantitative modelling of the system's metrics. Hence we do not focus on evaluating the absolute values. The goal of the study is to demonstrate the flexibility of the method w.r.t. a heterogeneous platform and a variety of tasks.

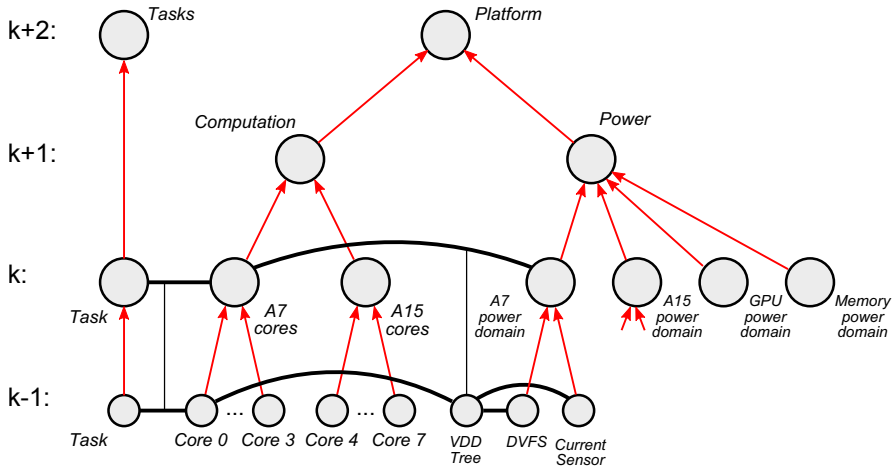

Figure 5. Order Graph model of running tasks on Odriod XU3 platform (some horizontal dependencies are omitted).

IV. Case Study

A. Platform Description

The Odroid XU3 board [1] is a small Octa-Core computing device implemented on energy-efficient hardware. The board can run different versions of OS, for example Ubuntu 14.04 or Android 4.4.

The main component of Odroid XU3 is the 28nm Application Processor Exynos 5422. This System-on-Chip is based on the ARM big.LITTLE architecture [9] and consists of a high performance Cortex-A15 quad core processor block, a low power Cortex-A7 quad core block, a Mali-T628 GPU and 2GB DRAM LPDDR3. The board contains four real time current sensors allowing the measurement of power consumption on the four separate power domains: big CPU, little CPU, GPU and DRAM. There are also four temperature sensors for the A15 processors and one for the GPU.

On the Odroid, for each power domain, the supply voltage (Vdd) and clock frequency can be tuned through a number of pre-set pairs of values, allowing dynamic frequency scaling (DFS) when the frequency is between 200MHz and 800MHz (the Vdd stays constant in this region) and dynamic voltage and frequency scaling (DVFS) [5], [11] when the frequency is 800MHz and above.

B. Platform Model in Order Graphs

In this case study we focus on modelling power consumption of the platform. Two major contributors are task affinities (which task runs on which core) and DVFS. Figure 5 shows the OG model of the system. At the higher levels of abstraction, the system is represented as a set of tasks running on a platform, which in turn consists of a computation component and a power component. The computation resource is provided by A7 and A15 cores, which appear in the lower orders, and the power resource is divided into four power domains, as described in Section IV-A. For clarity, some of the horizontal edges on this diagram are hidden: every core is actually connected to the corresponding Vdd tree and to the task node, etc.

C. Platform Model Components

SANs are an extension to Generalised Stochastic Petri Nets (GSPNs) and a more expressive representation language. The SANs formalism provides a general way of specifying the enabling of an activity or transition, a general way of specifying a completion (firing) rule, a method of representing zero-time events (hence including deterministic as well as stochastic behaviours), a method of representing probabilistic choice in addition to probabilistic delay provided by GSPNs. It also provides state-dependent parametric values and general delay distributions on activities.

With the Odroid platform, the major controls available for runtime power management are the DFS and DVFS of the core blocks (power domains), the activation and inactivation of individual cores, and the mapping of specific threads or tasks to individual cores. Figure 6 shows several possible ways of modelling these choice-based decisions. The task scheduling models describe an environment where tasks are organised into three queues, one going to the A15 processors, one going to the A7 processors, and one with non-deterministic designations. The algorithm in these models sets tasks to either the A15 or the A7 queue. This is done by specifying the logic for the output gates to decrement and increment the task queue markings accordingly when a transition fires. For instance, if transition A15 fires, one of the tasks in one of the other queues is moved to the A15 queue. The two models have different levels of fidelity in their representation; Figure 6(b) is a more deterministic case; Figure 6(a) is a more probabilistic case. The modelling and analysis costs/efforts of these models are related to their representational fidelity. The DVFS model in Figure 6(c) is the most probabilistic, has the least fidelity, and is the easiest to use, but higher fidelity versions up to fully deterministic can also be constructed. In these models, the task scheduling and DVFS transitions are assumed to be triggered by other sub-nets representing the controllers, which are not included here.

The other crucial issue to be modelled for this system, when we talk about system power consumption, is processing, i.e. the execution of threads/tasks in the cores. The fundamental

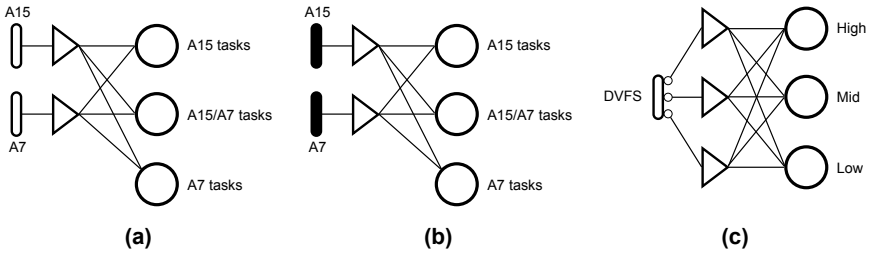

Figure 6. SANs models for stochastic (a) and deterministic (b) affinity, and DVFS (c).

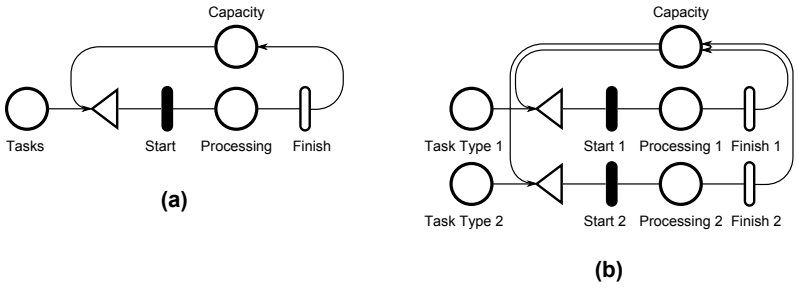

Figure 7. SANs models for task execution.

processing element model is shown in Figure 7(a). Here the place Capacity represents the unused capacity of a processing element (e.g. a core), and the place Processing represents the current number of tasks being executed in the core. If it is a single core, the sum of markings of these two places represents the pipeline depth or multi-threading capability of the core. If there are multiple cores in this model, the sum of markings represents the entire block's multi-threading capability.

Different levels of fidelity are possible with this representation. For instance the degree of probabilistic vs. deterministic can be tuned for a more or less fuzzy representation. We may decide to model part of a core (i.e. a multiplier), an entire single core, a core-block, or the entire Odroid chip with one of these sub-nets. When setting up a more detailed model with higher fidelity, we may need to distinguish how a processing element behaves with different types of tasks (see Figure 7(b)), as shown in subsequent sections the Odroid cores consume different amounts of power when dealing with different tasks. Another power-related issue in terms of processing is the workload or CPU utilisation when running a task on a CPU, which could be a number between 0 and 100%. Deterministic modelling of workload can be done similarly to what's done for tasks in Figure 7(b), with the workload resolution determining how many copies of the processing sub-net to use. For example, a resolution of 20% will result in 5 such sub-nets each representing 0-20%, 20-40%, etc. With more fuzzy representations, all such issues may be covered by probabilities using rates in the transitions.

Once a cut has been determined using the OG model, a flat SANs model covering the entire system can be made with different levels of fidelity for different parts. This will be a flat model with power-proportional fidelity and effort.

D. Model Characterisation Experiments

Experiments with the Odroid platform were carried out in order to understand the power consumption under different operation frequencies and voltages. The low-power A7 quad core block can scale its frequencies from 200MHz to 1400MHz, whilst the performance-oriented Cortex A15 block has a range of frequenciy from 200MHz to 2000MHZ. The frequency of each block can be changed independently using OS commands and the system scales the operating voltage of the block to fit the chosen frequency. The on-chip sensors allow voltage, current and power for each processor to be measured in real time.

In our characterisation experiments, firstly the above parameters were measured without any additional workload, with only the OS running. Then the same parameters were measured for each core with application threads running. We experimented with the typical Linux stress task, i.e. running square root calculations repeatedly, and in addition, other computations including logarithm calculations and the four arithmetic operations. We also covered different levels of workload.

Another important experiment is the measurement of the same parameters with some of the cores in each block disabled: Odroid allows from one to four of the A15 being disabled and from one to three of the A7 to be disabled. At least one A7 must be running for the OS to be alive.

In these experiments, it was observed that an A15 consumes four times or more power than an A7 when both are running at the same frequency, up to an order of magnitude more power when both are running at the same voltage. Figure 8 shows the relationship between power consumption and the execution time for the two types of cores on the average running a range of different types of tasks.

These radically different performance and power figures, and their complex relations to the different tasks being ex-

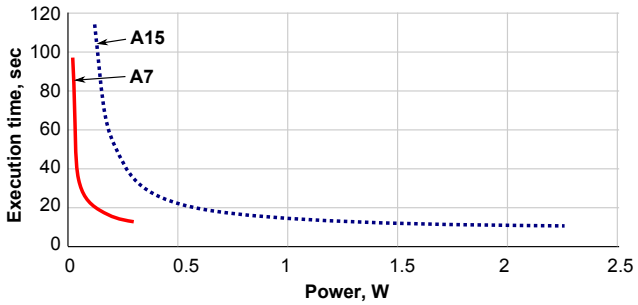

Figure 8. Measured power to execution time.

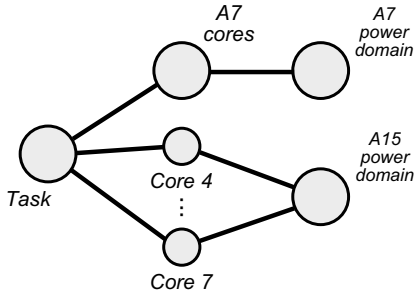

Figure 9. Proposed cross-layer cut for power-proportional modelling of Odriod XU3.

ecuted in a core, validate the approach promoted in this paper. For instance, when certain tasks are mapped to the A7 block, because of the relatively light power demand of these cores we may be able to afford to model such processing with less fidelity, i.e. using a more probabilistic model and/or using a more structurally fuzzy model. For instance, when the A15 block is also running, it may be a good idea to not represent individual A7 cores but to cover the entire A7 block with a single model of the type in Figure 7(a).

E. Power proportional model sizes

Using the modelling flow presented in Section III and based on experimental data from the Odroid, presented earlier in this section, for a certain modelling fidelity we may need to represent each A15 core with a model of the type in Figure 7(b), with multiple types of tasks – e.g., CPU heavy and memory heavy, and many levels of DVFS and workload resolutions. For the same level of fidelity, we can represent the entire A7 block with a single sub-net of the type in Figure 7(a) without task, DVFS and workload differentiation.

The corresponding OG cut is shown in Figure 9.

Power proportional cuts through the model space usually result in models whose sizes are optimal for studying power, in the sense that the resolution or fidelity of power as a parameter is constant through the model. In other words, a power proportional cut for a specific power representational fidelity gives the smallest possible model for that degree of fidelity. Other representations away from this cut will inevitably result in certain parts showing an unnecessarily higher degree of fidelity leading, usually, to higher degrees of representational complexity.

One of the generally accepted metrics of model size and therefore modelling effort and the effort of using models is the size of the state space of the model. For instance, one of our envisaged applications of our modelling method is the design and analysis of runtime parametric management algorithms or machines, e.g. runtime power management for mobile and embedded systems. For such management or control schemes, more sophistication is usually needed to achieve better results. Naïve examples that are widely available in the public and commercial domains, such as such Linux/Android power governors as `ondemand`, usually assume very simplistic plant models and rely on feedbacks to achieve some degree of effectiveness, which is almost never optimal. More sophisticated algorithms such as those based on learning and those providing a degree of adaptation can almost always provide better results than the standard governors, but inevitably require better plant models. On the other hand, most computer system control algorithm designers are most comfortable with thinking of the plant as a state machine. And many types of parametric management algorithms, e.g. learning and model adaptive schemes, depend on a state space representation of the plant being available [6], [18], [7], [13]. The size of the state space of a model, therefore, is directly relevant for this type of model usage.

The example architecture of the type seen in the big.LITTLE Exynos chip featured in the Odroid system consists of N power domains. The k-th power domain, $0 \leq k < N$, has d_k DVFS points (pre-set pairs of Vdd and clock frequency values) and c_k processing cores of the same type, each of which supports a workload between 0 and 100% running t_k types of tasks. For such a system and for power studies, the fundamental state element is 'a particular core in a particular power domain is running a particular type of task at a certain workload'. Usually the parametric representational fidelity requirement dictates the granularity of workload representation, which should be constant within each individual power domain, as there is no intra-domain core heterogeneity. This leads to each of the c_k cores having w_k workload points and t_k types of tasks. The size of the state space S of a cut model for such a system as the controlled plant, of the type described in the previous sections, is therefore:

$$|S| = \prod_{k=0}^{N-1} d_k c_k w_k t_k.$$

For the particular example of managing the Odroid's 8 general processing cores, reducing the representation of the A7 cores to a single execution sub-net model, as in Figure 9, produces 4 times reduction of the state space (the state space is reduced to 25% of the original full model). Reducing the workload resolution of the A7 cores to equal power fidelity of the A15 cores produces a similar reduction of the state space (i.e. the same power fidelity results in workload resolutions of $w_{A15} \geq 4$ and $w_{A7} = 1$). The same is true for DVFS resolution. There are 20 DVFS points for the A15 power domain and 15 for the A7 power domain in the Odroid. Maintaining the same power fidelity, there is no need to represent all 15 of the A7 DVFS points. Even if we represent all 20 DVFS points for the A15 domain, a maximum of 5 DVFS bands are needed for the A7 domain in the model. And with the simplest possible

task type differentiation, CPU-heavy and memory-heavy, for the A15 processors, the A7 domain would have no need for task type differentiation. Given all these considerations and without going into any other more sophisticated elaborations, a 128 times reduction of the state space (new state space size = 1/128 or 0.8% of the original) can be obtained by exploiting the knowledge that each A15 core is as significant as the entire A7 block in terms of power. This kind of state space reduction may result in qualitative differences in the sophistication of the runtime management scheme given any constant overhead budget for the management, or it can be used to reduce the management overhead whilst maintaining the same degree of management effectiveness. This modelling method provides more opportunities for runtime tuning and adaptation because cuts may be allowed to dynamically change during run time. For instance, if it is found that no A15 core is active and the entire A15 block is shut down, power fidelity may be improved by representing the A7s individually by adopting a different cut. This can be necessary because A15 total shutdown in may indicate that the system is running in low power or even survival modes and during these modes what are regarded as small amounts of power during normal operation become significant. This should lead to a higher degree of fidelity in representing the quantity of power in the runtime model. On the other hand, if the A15 total shutdown is purely a result of workload reduction, the previous coarse A7 block cut should be entirely satisfactory. The facility of layer-crossing cuts provides additional flexibility for model tuning and adaptation.

V. Conclusions

This paper, to our knowledge for the first time, formally recognises the need for parametric significance-driven design, where modelling fidelity and hence effort are proportional to design parameters. This concept is supported by a parametric significance-driven modelling approach to complex systems, illustrated by using power as an example metric in this paper, resulting in the reported modelling flow which has power-proportional fidelity.

This method is centred on Order Graphs, a new formalism with facilities for independent vertical zooming among different parts of a model, and the straightforward exploration and discovery of appropriate parametric-proportional cuts. These cuts are then very suitable for exploration, reasoning and analysis with established flat qualitative and quantitative representation methods. Here SANs are used as an example for this type of exploration. Experiments with a heterogeneous system with multiple cores and power domains as well as different types of computation tasks help validate and further motivate the approach. Analysis showed that parametric-proportional cuts can reduce the size of models radically when system heterogeneity can be exploited.

The future work would focus on two aspects. Theoretically, the OG modelling method will be expanded beyond the static knowledge stage into dynamic progression semantics. Practically, the proposed method will be applied to the development of an intelligent runtime parametric control scheme for heterogeneous many-core systems.

Acknowledgement This work is supported by EPSRC grant EP/K034448/1.

References

[1] Odroid XU3. http://www.hardkernel.com/main/products.

[2] G. Balbo. *Formal Methods for Performance Evaluation*, volume LNCS4486, chapter Introduction to Generalized Stochastic Petri Nets, pages 83–131. Springer, 2007.

[3] S. Borkar. Thousand core chips: A technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 746–749, New York, NY, USA, 2007. ACM.

[4] Yitzhak Brave and Dominique Bonvin. A minimally restrictive policy for deadlock avoidance in a class of fms. In Silvano Balemi, Petr Kozák, and Rein Smedinga, editors, *Discrete Event Systems: Modeling and Control*, volume 13 of *Progress in Systems and Control Theory*, pages 57–69. Birkhäuser Basel, 1993.

[5] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low power CMOS digital design. *IEEE Journal of Solid State Circuits*, 27:473–484, 1995.

[6] Anup K. Das et al. Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems. June 2014.

[7] Anup K. Das et al. Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems. In *Conference on Design, Automation & Test in Europe*, March 2015.

[8] A. Ehrenfeucht and G. Rozenberg. Zoom structures and reaction systems yield exploration systems. In *IJFCS*, pages 275–306, 2014.

[9] P. Greenhalgh. *big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7 – Improving Energy Efficiency in High-Performance Mobile Platforms*. ARM, 2011. White Paper.

[10] B. Kumar and E. S. Davidson. Computer system design using a hierarchical approach to performance evaluation. *Commun. ACM*, 23(9):511–521, September 1980.

[11] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proc. of 2010 Intern. Conf. on Power Aware Computing and Systems*, HotPower'10, USA, 2010.

[12] Yves Lhuillier et al. HARS: A hardware-assisted runtime software for embedded many-core architectures. *ACM Trans. Embed. Comput. Syst.*, 13(3s):102:1–102:25, March 2014.

[13] Luis Alfonso Maeda-Nunez et al. PoGo: an application-specific adaptive energy minimisation approach for embedded systems. In *HiPEAC Workshop on Energy Efficiency with Heterogenous Computing (EEHCO)*. HiPEAC, January 2015.

[14] R. Murphy, S. Carter, M. Ornelas, and S. Deshpande. System and method for dynamic resource reconfiguration using a dependency graph, September 9 2004. US Patent App. 10/382,427.

[15] A. Rafiev et al. Studying the interplay of concurrency, performance, energy and reliability with ArchOn – an architecture-open resource-driven cross-layer modelling framework. In *Proc. to ACSD*, 2014.

[16] A. Rafiev et al. Power-proportional modelling fidelity. In *Proc to Model-Implementation Fidelity (MiFi)*, 2015.

[17] W.H. Sanders and J.F. Meyer. *Lectures on Formal Methods and Performance Analysis*, volume LNCS2090, chapter Introduction to Generalized Stochastic Petri Nets, pages 315–343. Springer, 2001.

[18] A. Suardi, S. Longo, E.C. Kerrigan, , and G.A. Constantinides. Robust explicit MPC design under finite precision arithmetic. In *Proc. to IFAC*, 2014.

[19] Bo Wang et al. End-to-end power estimation for heterogeneous cellular LTE SoCs in early design phases. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on*, pages 1–8, Sept 2014.

[20] Fei Xia, Alexandre Yakovlev, Ian G. Clark, and Delong Shang. Data communication in systems with heterogeneous timing. *IEEE Micro*, 22(6):58–69, 2002.